gdb, valgrind, testimine Võistlus programmeerimise kontekstis

Oliver Nisumaa

11. märts 2017. a.

võimalikud vead

- Syntax error
- Runtime: assertion failed, exception, segmentation fault
- Vale vastus näidistestil
- Vale vastus sügaval lõpptestides

assert

C funktsioon, mille saab `<cassert>` headerist.

```
void assert (int expression);
```

Kui `expression` väärtus on 0 ehk false, siis programm jookseb kokku spetsiifilise veateatega

```
Assertion failed: expression, file filename, line line number
```

GDB-ga saab hakkata vastavalt realt debugima.

assert

Aeglustab märgatavalt programmi. Kui lisada `#define NDEBUG`, siis asserte(ja avaldisi nende sees) ei kompileerita. GCC-ga saab ajutisel kompileeerida ka `g++ -DNDEBUG ...`

assert

Kasutamise olukorrad:

- Kahtlete, kas mingi osa koodist ikka teeb seda, mida vaja.
- Kahtlete, kas mingi matemaatiline tulemus ikka kehtib.
- Kahtlete, kas mingi STL funktsioon ikka töötab nii, nagu arvate.

Kontrollimise põhjalikkus sõltub ka sellest, kui lihtne mingit asja kontrollida on. Enamasti kontrollite vaid osaliselt.
Asserte võib lisada nii koodi kirjutamise ajal kui ka pärast debugides.
Enamus võistlustel saab assertidega koodi esitamisega uurida, milline osa koodist valesti töötab.

valgrind

Võimekas, väga keeruliseid asju oskav tööriist. Programm pannakse käima valgrindi poolt implementeeritud virtuaalsel protsesoril, valgrind saab programmi tegevust põhjalikult analüüsida ja kontrollida. Valgrindiga jookseb programm enamasti 10 kuni 100 korda aeglasemalt, kui gdb-ga. Valgrindil on palju tööriistu(Teoreetiliselt ka SGcheck), võistlusel debuggimisest on kasulik ainult memcheck. Memcheck peab arvet, millist osa mälust on programmille allokeeritud, millline osa sellest defineeritud(ei ole ise ega sõltu initialiseerimata väärtustest). Valgrind annab teile teada KÕIGIST kordadest, kui teie programm näpib mälu, mis ei ole talle antud. Valgrind annab teile teada, kui te initialiseerimata väärtused mõjutavad programmi käiku või väljundit. Lõpuks väljastab valgrind info mälulekkete kohta. Mälu lekkete vältimine on praktikas väga oluline, kuid võistlustel need enamasti otseseid probleeme ei tekita.

valgrind

Kõige olulisem osa on lubamatu mälu näppimine. Viga valgrindi väljundist üles leida võib olla raske. Õnnneks saab valgrindile gdb külge panna.

- Kompileerige `vectortest.cpp`(Koos debuggimisümbolitega, ilma optimeerimiseta).
- Jooksutage seda mõned korrad.
- Pange see korra ka gdb-ga korra käima(lihtsalt run, ei mingeid breakpointe).
- Jooksutage nüüd `valgrind exe`, memcheck on valgrindi vaikimisi töörist.
- Tehke nüüd kaks terminali lahti, liikuge mõlemas programmiga samasse kausta. Ühes terminalis tehke käsk `valgrind --vgdb-error=0 exe`. Teises terminalis tehke `gdb exe`. Sisestage gdb-sse valgrindi poolt öldud käsk.
- Nüüd on ühendus loodud, `exe` pole veel käima pandud, iga `continue` käigu peale liigutakse järgmise valgrindi poolt avastatud vea juurde. Saate kasutada sama funktsionaalsust, mis tavalise gdb sessiooni ajal(ka breakpointe ja watchpointe lisada). `exe` stdin ja stdout käivad

valgrind

Kogu selle äärmiselt tülika kahe terminaaliga jamamise võiks bash scripti `valgdb.sh` automatiseerida

```
#!/bin/bash

prog="$1"
if (($# > 1)); then
        valgrind --vgdb-error=0 $prog < $2 &
else
        valgrind --vgdb-error=0 $prog &
fi
valgrind_pid=$!
gdb --eval-command=
        "target remote | vgdb
        --pid=$valgrind_pid"
        $prog
kill -9 $valgrind_pid >/dev/null 2>&1
```

valgrind

Valgrind on väga täpne programmide analüüsimises.
Analüüsige `valgrind_smartness.cpp`
Teil on ka `valgdbadv.sh`, mis lubab tööriista valida.

Kui viga ilmub alles serveris . . .

- See võib tähendada, et optimaalne on kirjutada automaattester.
- Hea automaattesteri kirjutamine ei pruugi olla võimalik.
- Automaattesteri kirjutamine võib olla väga erineva raskusega.
- Täielik automaatester koosneb
  - testitavast programmist,
  - lihtsast, lollikindlast näidisprogrammist(edaspidi loll programm).
  - testide generaatorist
  - testerist, mis liimib programmid kokku.

Mõningad näidisjuhud programmide testitavusest

1. OML-i võistlus A
2. OML-i võistlus B
3. Lõppvoor edasijõudnud 1. ülesanne(Gümnaasium 2.).
4. OML-i võistlus C
5. Lõppvoor edasijõudnud 2. ülesanne(Gümnaasium 3.).
6. Lõppvoor edasijõudnud 3.
7. http://codeforces.com/problemset/problem/768/F
8. http://codeforces.com/problemset/problem/768/E

Tester, generaator c++ga

C omab system käsku.

```
int system (const char* command);
```

Tagastab jooksutatava käsu väljumiskoodi. C++ stringe saab muuta C-strgideks `c_str()` meetodiga.
Linuxis on käsk `diff file1 file2`, mis prindib failide erinevuse kirjelduse, kui väljundid erinesid.
`diff` väljumiskood on 0, kui erinevust ei leitud ja 1, kui leiti.
Vaadake kaustas safety(ülesande nimi) olevat testerit.

Lõpp